Harmonia: Balancing Compute and Memory Power in High-Performance GPUs

Indrani Paul*† Wei Huang* Manish Arora*‡ Sudhakar Yalamanchili†
*AMD Research; †Georgia Institute of Technology; ‡University of California, San Diego
{indrani.paul,wein.huang,manish.arora}@amd.com; sudha@ece.gatech.edu

Abstract

In this paper, we address the problem of efficiently managing the relative power demands of a high-performance GPU and its memory subsystem. We develop a management approach that dynamically tunes the hardware operating configurations to maintain balance between the power dissipated in compute versus memory access across GPGPU application phases. Our goal is to reduce power with minimal performance degradation.

Accordingly, we construct predictors that assess the online sensitivity of applications to three hardware tunables—compute frequency, number of active compute units, and memory bandwidth. Using these sensitivity predictors, we propose a two-level coordinated power management scheme, Harmonia, which coordinates the hardware power states of the GPU and the memory system. Through hardware measurements on a commodity GPU, we evaluate Harmonia against a state-of-the-practice commodity GPU power management scheme, as well as an oracle scheme. Results show that Harmonia improves measured energy-delay squared (ED^2) by up to 36% (12% on average) with negligible performance loss across representative GPGPU workloads, and on an average is within 3% of the oracle scheme.

1. Introduction

Graphics processing units (GPUs) are now commonly used for data parallel applications that do not fit into the traditional graphics space. They have been shown to provide significant improvements in power efficiency and performance efficiency for many classes of applications [3, 28, 21]. However, while compute has been a major consumer of power in such systems, moving forward we see that the power spent in the memory system and in data movement will begin to become major, and sometimes dominant, components of platform power [29, 47]. For example, Figure 1 illustrates the power distribution in an AMD Radeon™ HD7970 discrete GPU card (dGPU) executing a memory intensive workload *XSBench* [22]. This emerging redistribution of power consumption between compute and memory must operate under a fixed board level power and thermal envelope, while with the advent of on-package DRAM (e.g., die stacks and EDRAM) [43, 19, 26, 38], they must share an even tighter package power and thermal envelope. Therefore we argue that effective dynamic power redistribution between compute and memory will be key to energy and power efficiency for future high-performance GPUs.

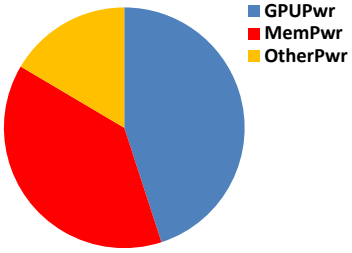

Figure 1: Power breakdown in a typical modern discrete GPU card for a memory-intensive workload.

The underlying principle of our approach is to match the relative power consumption of GPU cores and the memory system, with the relative compute and memory demands of the applications. For example, the ops/byte value of an application (number of compute operations per byte of memory data transfer) represents the relative demand placed on the GPU cores and the memory system. Hardware tunables such as the number of parallel cores, core operating frequency, and the memory bandwidth collectively capture the relative time and power cost of performing operations versus memory accesses in the hardware platform. Ideally, the relative ops/byte demand of the applications matches the relative time and power costs of compute and memory hardware of the platform and we have a perfectly balanced system [9, 51], without wasted power and/or unexploited performance opportunities. In reality, application behavior is time-varying, and the ops/byte costs of the platform depend on the values of the hardware tunables. For example, we studied the behavior of *Graph500* [37] running on an AMD[1] Radeon HD7970 GPU card with GDDR5 memory [35]. Its ops/byte varies from lows of 0.64 ops/byte to bursts of 264 ops/byte. The high demand on ops/byte of the application implies the memory system can be run at lower speeds relative to compute with negligible performance degradation but significantly lower power. Hence, to retain the most power efficient operation, we need a runtime power management infrastructure that can coordinate power states of the processor (GPU) and the off-chip memory system so that they are in balance, or in "harmony".

ISCA'15, June 13-17, 2015, Portland, OR USA

DOI: http://dx.doi.org/10.1145/2749469.2750404

[1] AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

In this paper, we propose *Harmonia*, a runtime scheme that adjusts the hardware tunables on a state-of-the-art, high-performance discrete GPU to balance the power in the memory system and GPU cores to match the desired ops/byte characteristics of a high performance computing (HPC) application. We show how such a balance can reduce overall platform power with little compromise in performance. Our focus is on the HPC domain where applications are characterized by relatively uncompromising demands for execution time performance, thereby placing stringent demands on improvements in power and energy efficiency.

Specifically, this paper makes the following contributions:

- Through measurements on a modern GPU, we provide a characterization of representative high-performance and scientific computing applications with respect to their: i) operation intensity, and ii) performance sensitivity to three hardware tunables—the number of GPU compute units (CU), CU frequency, and memory bandwidth.
- Based on this characterization, we derive online models that predict performance sensitivity of application kernels to each of the preceding three hardware tunables.
- We propose a coordinated two-level power management scheme, *Harmonia*, to tune platform balance between compute throughput and memory bandwidth by: i) a coarse-grain adjustment of the GPU and the memory power states based on online sensitivity prediction, ii) followed by fine-grain tuning through close-loop performance feedback.
- Using measurements from an implementation on commodity hardware, we compare Harmonia to a commercial, state-of-the-practice power management algorithm, demonstrating that up to 36% (average of 12%) improvements in energy-delay-squared product (ED^2) are feasible with minimal sacrifices in performance. In addition, we also show that Harmonia achieves to within 3% of an oracle scheme.

The following section describes the state-of-the-art platform used in the measurements of the paper. The remainder of the paper successively presents a detailed characterization of the platform, the performance sensitivity model, the Harmonia algorithm, and insights from the evaluation.

2. Background and Baseline System

A GPU is a data parallel execution engine consisting of collections of simple execution units or Arithmetic Logic units (ALUs), operating under the control of a single instruction multiple data (SIMD) stream partitions. There are modern programming languages based on the bulk synchronous parallel (BSP) model, such as OpenCL and CUDA, taking advantage of massively parallel GPU architectures. This paper utilizes the OpenCL terminology, although the concepts are applicable to analogous elements of CUDA.

2.1. GPU Concurrency Model

A host program launches a kernel consisting of a 2D/3D grid of workgroups where each workgroup is comprised of a block of workitems (threads). Workgroups share a block of local data storage (LDS) and vector and scalar general purpose registers (VGPR and SGPR). Workitems within a workgroup are also grouped into sets of threads called wavefronts operating in lock step relative to each other. A wavefront is the basic unit of hardware scheduling. However, there are resources that are shared among workgroups. Conflicting resource demands and sharing in part govern the number of in-flight wavefronts and hence concurrent execution.

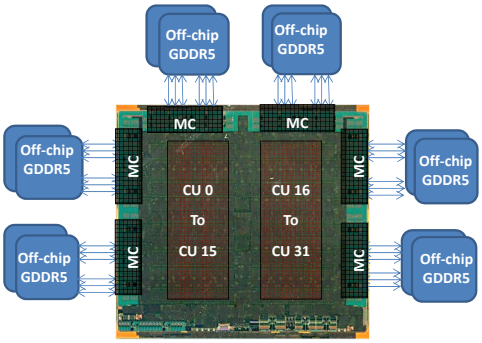

Figure 2: AMD HD7970 GPU Architecture [35].

2.2. GPU Hardware Architecture

We use the AMD Radeon HD 7970 system as our test bed [35]. This platform is one of the "Southern Island" families of AMD graphics processors, and is illustrated in Figure 2. It features the AMD Graphics Core Next (GCN) architecture and is paired with 3GB of GDDR5 memory organized using a set of six 64-bit dual channel memory controllers (MC) with maximum bandwidth of 264GB/s. The processor contains up to 32 compute units or CUs with four SIMD vector units in each CU. There are 16 processing elements (PE) per vector unit, called ALUs, resulting in a single precision fused multiply-accumulate (FMAC) compute throughput of about 4096 GFLOPS. Each CU contains a single instruction cache, a scalar data cache, a 16KB L1 data cache and a 64KB local data share (LDS) or software managed scratchpad. All CUs share a single 768KB L2 cache. All CUs in the GPU share a common frequency domain and a voltage plane.

2.3. GPU Power Management

The HD7970 uses AMD PowerTune technology [1] to optimize performance for thermal design power (TDP)-constrained scenarios. The GPU adjusts power between the DPM0, DPM1 and DPM2 power states shown in Table 1, based on power and thermal headroom availability. It also allows for a boost state of 1GHz at 1.19V supply voltage when there is headroom. This works well for managing compute power. However, very little power management exists for off-chip memory which shares the same platform-level power budget on current GPUs, and same on-die power and thermal envelope in future GPUs that use 3D die-stacking [39].

2.4. Memory Power Management

As can be seen in Figure 1, memory is a significant power consumer in the GPU. One way to change memory power is by dynamically adjusting the memory bus frequency, which controls the memory controller, GDDR PHY, and the DRAM devices. DRAM power can be further broken down into background, activation/pre-charge, read-write, and termina-

tion power. Changing memory bus frequency has a different impact on each of these components. Lowering bus frequency lowers background and PLL power, as well as memory controller and PHY power. On the other hand, it can increase read/write and termination energy due to longer intervals between array accesses. Further, if frequency is slowed down to a point where memory latency can no longer be hidden through thread-level parallelism in the GPU, it can hurt performance significantly and increase the overall energy consumption of the platform. In this paper, due to hardware limitation, we use only memory channel (i.e., bus) frequency as the knob to manage memory power and memory bandwidth.

GPU DVFS state	Freq (MHz)	Voltage (V)
DPM0	300	0.85
DPM1	500	0.95
DPM2	925	1.17

Table 1: AMD HD7970 GPU DVFS table.

3. Motivation and Opportunities

The compute throughput of the GPU is determined by the number of active CUs and their operating frequency. Similarly, memory bandwidth is determined by the frequency of the memory bus. Our goal is to strike the right balance between the settings for compute throughput and memory bandwidth as determined by application characteristics. Towards this end, this section presents a characterization of the relationship between application behaviors and the settings for compute throughput and memory bandwidth.

3.1. Experimental Methodology and Terminology

In the AMD Radeon HD 7970, the number of active CUs is adjustable from 4 to 32, and the CU frequency can be varied from 300MHz to 1GHz, in steps of 100MHz. We call a specific setting of the CU count and CU frequency as the *compute configuration*. Memory bandwidth can be varied from 90GB/s (at 475MHz bus frequency) to 264GB/s (at 1375MHz bus frequency), in steps of 30GB/s (150MHz). A specific setting is called the *memory configuration*. The total number of combinations of compute and memory configurations is approximately 450. Each combination reflects a specific value of ops/byte that the platform hardware can deliver. It also reflects a specific balance between power devoted to compute and memory. Significant imbalance between demanded ops/byte of the application and what the platform delivers can result in longer execution time and/or energy inefficiencies.

3.2. Performance-Power Scaling and Hardware Balance

Figure 3 shows the normalized measured performance of three different applications: a) *MaxFlops*, b) *DeviceMemory*, and c) *LUD*. Among them, *MaxFlops* and *DeviceMemory* are benchmarks from the SHOC suite [12] that are commonly used in the GPGPU community to stress the GPU hardware against its compute and memory limits, respectively. *LUD* is a representative scientific application from the Rodinia benchmark suite [6, 7] that performs matrix decomposition. The X-axis shows the ops/byte provided by the hardware. Each curve in the figures corresponds to a fixed memory configuration. Each point on the curve is a different compute configuration with increasing CU frequency and number of CUs as we move to the right (i.e., increasing ops/byte of the platform). The Y-axis shows performance (i.e., 1/execution time). Both the X and Y axes are normalized to those of a minimum hardware configuration with 4 active CUs, 300MHz compute frequency and 90GB/s memory bandwidth.

MaxFlops is a compute-bound application. As we can see from Figure 3(a), increasing compute throughput results in linear increase in performance for a fixed memory bandwidth. Also, for the same compute-to-memory bandwidth ratio in the platform (i.e. same ops/byte value on the x-axis), higher available memory bandwidth means higher available compute throughput and hence higher performance for this benchmark. However, it is clear that maximum performance (at 27 normalized performance on Y-axis) is achieved at multiple memory configurations. All these points are at the same compute configuration—maximum 32 CUs and maximum 1GHz compute frequency. However, the most energy-efficient point is the rightmost point at 27 normalized ops/byte of x-axis, which corresponds to the lowest memory bandwidth. This is because *MaxFlops* is not memory sensitive—running at the lowest memory bandwidth does not hurt performance, but significantly improves energy efficiency.

Now consider the memory-bound application *DeviceMemory* in Figure 3(b). We observe that for each value of memory bandwidth, increase in compute throughput does not lead to improved performance beyond a hardware ops/byte of around 4.0. This is because performance is eventually limited by the memory bandwidth as we increase compute throughput by increasing the number of CUs and CU frequency. Hardware configurations with normalized ops/byte of $\sim$4.0 are balanced configurations where compute throughput just saturates the available memory bandwidth. Each memory configuration has a different balance point (the knee of the curve) corresponding to a specific compute configuration. The optimization problem is the selection of the specific balance point that maximizes power and energy efficiencies with minimal impact on performance. Any other combination of compute and memory configurations either wastes power and/or leaves additional performance gains unexploited.

Finally, in Figure 3(c) we show the behavior of *LUD*. The application may be compute-bound or memory-bound depending on the choice of compute and memory configurations. For higher values of memory bandwidth, the application remains compute bound across all configurations. For such applications, the best hardware balance point corresponds to the configuration that is the highest and rightmost. For *LUD*, this is achieved when normalized hardware ops/byte is at around 15, where compute throughput most effectively matches memory bandwidth demands.

In general, the optimal hardware balance point varies across applications and application phases. It also varies across different hardware platforms. The techniques we propose in this paper dynamically adapt the hardware platform characteristics to match the runtime ops/byte behavior of the

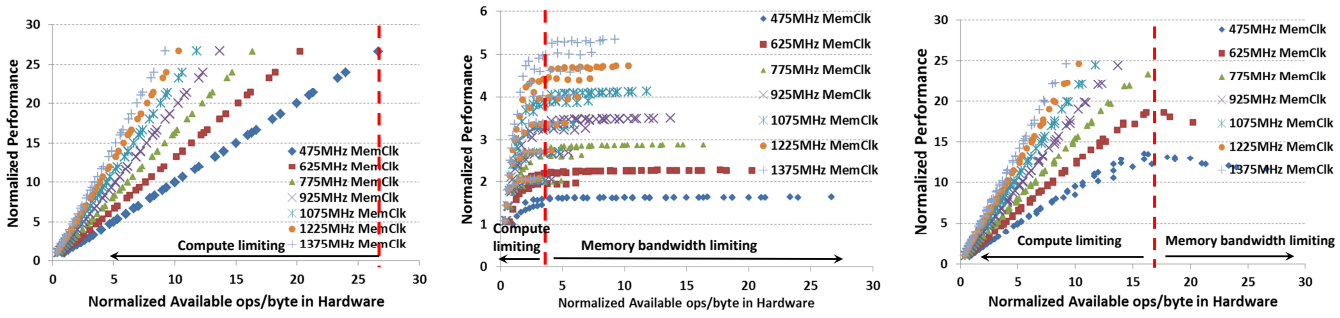

Figure 3: Hardware balance points for: (a) *MaxFlops*, (b) *DeviceMemory*, and (c) *LUD*.

applications. The result is reduction of unnecessary power (i.e., power that has little impact on performance).

3.3. Power Reduction Opportunities

We characterize the power reduction opportunities by examining the effect of changing the platform ops/byte by: i) changing the compute configuration for a fixed memory (bandwidth) configuration, and ii) changing the memory configuration for a fixed compute (throughput) configuration. We measure total power of the graphics card using the setup described in Section 6. Results are normalized to the power of a minimum hardware configuration with 4 active CUs, 300MHz compute frequency, and 90GB/s memory bandwidth.

In Figure 4, the X-axis indicates the available ops/byte in hardware under a constant maximum memory bandwidth of 264GB/s (i.e., fixed memory configuration). The Y-axis shows the impact of changes in compute configuration on overall board power for the memory-intensive *DeviceMemory* benchmark. Each set of points represents a CU count (4 to 32) and each point in a set shows increasing CU frequency. We see that normalized board power varies by about 70% across all compute configurations. This could be greater when operating at less power intensive memory configurations.

Figure 5 shows variation of board power across memory configurations for a maximum compute configuration (32 CUs and 1 GHz frequency) for the compute-intensive *MaxFlops* benchmark. Each point corresponds to one value of memory bandwidth. We see about a 10% power variation between operating at the lowest memory bus frequency of 475MHz (90GB/s) compared to the memory bus frequency of 1375MHz (264 GB/s). Note that the memory bandwidth variation is performed at a fixed voltage as the memory system voltage could not be controlled in our experimental setup. Therefore, the differences would actually be greater if we are able to scale memory bus voltage according to bus frequency. The potential percentage power savings would also be greater for less power intensive compute configurations.

3.4. Metrics

We note that HPC applications demand minimal degradations in execution time. Consequently, our goal is to minimize energy expenditure while keeping execution time constant (at best). This can be achieved by improving energy efficiency (ops/joule). Under a fixed execution time constraint it is

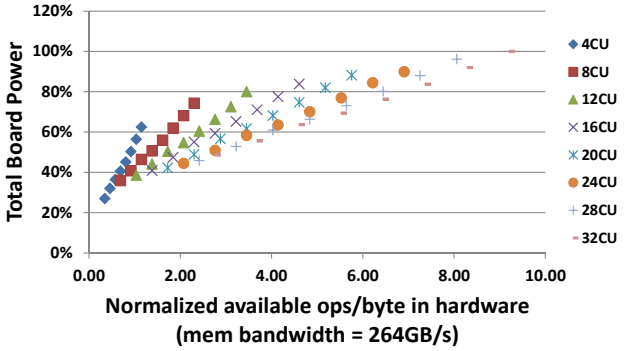

Figure 4: *DeviceMemory*'s GPU card power across compute configurations at constant 264GB/s memory bandwidth.

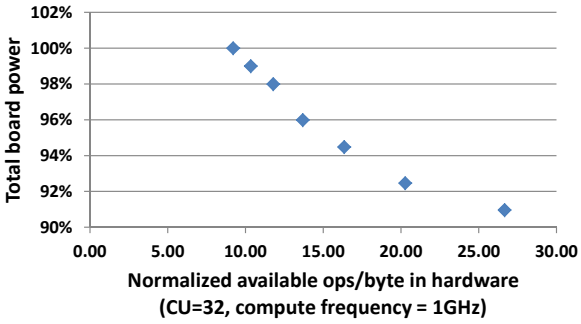

Figure 5: *MaxFlops*'s GPU card power across memory bandwidth configurations at 32CUs and 1GHz compute frequency.

equivalent to improving power efficiency. To capture this relative importance of *both* time and energy, we can utilize metrics of energy-delay (ED) and energy-delay square (ED^2). The latter in particular is commonly used in HPC application analysis [30, 49]. Here D means the actual time of kernel execution. In current technologies where leakage power can be a significant fraction of the total power, ED^2 captures effects of changes in both performance (Delay) and voltage.

Figure 6 shows the following analysis of the behavior of these metrics. We perform an exhaustive design space exploration across all 450 hardware configurations for *LUD* and *DeviceMemory* searching for the configurations that: i) minimize energy, ii) minimize ED^2, or iii) maximize performance, as indicated by the three bars in each group of columns. For each of these three configurations, we note the corresponding measured performance, energy, ED^2, and ED. All results

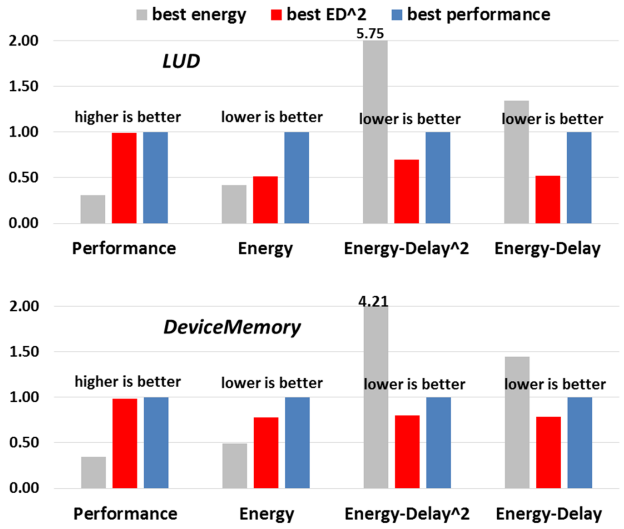

Figure 6: Performance, energy, energy-delay^2, and energy-delay comparisons for *LUD* and *DeviceMemory*. Energy optimality leads to significant performance impact.

are normalized relative to the best performing configuration. We find that the configuration optimizing for energy (1st bar) would result in 69% and 66% performance loss for *LUD* and *DeviceMemory*, respectively, compared to the best performing configuration (3rd bar). On the other hand, the configuration optimizing for ED^2 (2nd bar) has only 1% performance penalty, but still realizes 60% and 38% reduction in energy compared to the energy optimized case. For the rest of the paper, we use ED^2 as the main metric for evaluation motivated by its wide usage in HPC application analysis [30, 49] and note that using ED here yields similar conclusions.

3.5. Compute & Memory Bandwidth Sensitivity Analysis

The preceding subsections describe the scope of impact of hardware tunables on power and performance. To develop an online technique to effectively set these tunables we must understand the sensitivity of performance metrics to changes in values of these tunables. The sensitivity of performance to a hardware tunable is computed as the ratio of the relative change in the performance metric to the relative change in the corresponding values of the hardware tunable. Due to space limitation, we only present the most relevant data from a few representative applications and their kernels which have a variety of phases within a GPGPU application.

Kernel Occupancy and Latency Hiding. Kernel occupancy is a measure of concurrent execution and the utilization of the hardware resources (e.g., LDS, SGPRs and VGPRs), as discussed in Section 2.2). Figure 7 shows memory bandwidth sensitivity of kernel occupancy measured on HD7970 for *Sort.BottomScan* from the SHOC benchmark suite, and *CoMD.AdvanceVelocity* from the exascale proxy applications [22]. Here, *Sort.BottomScan* has a kernel occupancy of only 30%. The limiting factor is the number of VGPRs used. The VGPRs needed per wavefront is more than 25% (66) of the total number of available VGPRs (256), hence only 3 simultaneous wavefronts per SIMD unit (instead

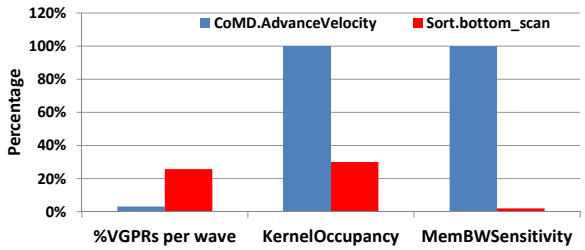

Figure 7: Effects of VGPR-caused kernel occupancy limitation.

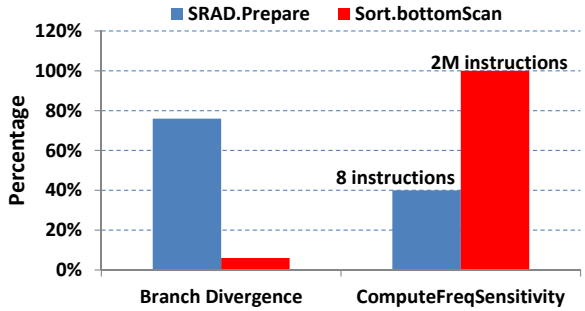

Figure 8: Impact on compute frequency sensitivity from load imbalance (branch divergence) and no. of instructions.

of a maximum 10) or 12 per CU can be in-flight concurrently. This leads to less sensitivity to memory bus frequency due to less degree of parallelism for *Sort.BottomScan*. On the other hand, *CoMD.AdvanceVelocity* has 100% kernel occupancy because the VGPR is not a limiting resource, leading to increased memory level parallelism and sensitivity to memory bandwidth.

Load Imbalance Due to Branch Divergence and Kernel Complexity. Control divergence causes thread serialization which can severely degrade performance. Prior works [36, 42] have shown that performance is sensitive to compute frequency for such workloads since it speeds up serial thread execution and shortens the overall execution time. However, frequency sensitivity cannot be inferred by branch divergence measures alone. Low divergence in large kernels can have a significant impact, while large divergence in small kernels (i.e., less number of dynamic instructions) may have little impact. Figure 8 shows compute frequency sensitivity for *SRAD.Prepare* and *Sort.BottomScan*, from Rodinia and SHOC benchmark suites respectively. The first set of bars indicates branch divergence and the second set indicates measured compute frequency sensitivity. While the *SRAD.Prepare* kernel has about 75% branch divergence, it has only 8 ALU instructions, making this kernel's impact on application performance less sensitive to compute frequency and more dominated by other overheads. However, *Sort.BottomScan* has only 6% branch divergence across over 2 million instructions, leading to significant thread serialization effects and load imbalances, and thus high sensitivity to compute frequency.

Architectural Clock Domains. Finally, we note that chip-scale global interactions between multiple clock domains can create non-obvious sensitivities. In our case, the GPU L2 cache (using the compute clock) and the on-chip memory controller (using the memory clock) are in different clock

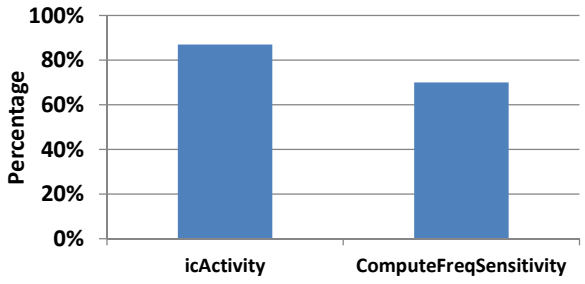

Figure 9: Impact of clock domains on compute frequency sensitivity for memory-intensive workloads.

domains. Reducing compute frequency reduces the rate at which requests are delivered from the L2 cache to the memory controller clock domain. For extremely memory-bound benchmarks with very poor L2 hit rates, slowing down the compute frequency can hurt overall performance. The left column in Figure 9 shows off-chip interconnect activity (icActivity) for *DeviceMemory*. This application has an ops/byte demand of ∼4.0 with poor cache hit rate in the L2, which would otherwise make this kernel memory bound. However, the right column in Figure 9 indicates its high sensitivity to compute frequency, especially when compute frequency is low since the effective bandwidth to the DRAM is reduced.

In summary, achieving hardware balance requires periodically assessing the sensitivity of performance to the hardware tunables accompanied by proportional changes to the values of the hardware tunables. The next section describes the development of sensitivity predictors for this purpose.

4. Compute and Memory Sensitivity Predictors

We develop models to predict the sensitivity of the application to compute throughput (set by active CU count and CU frequency) and memory bandwidth (set by memory frequency) configurations. The predictors are developed based on measurement data from a wide range of simple and complex applications with one or many kernels for a total of 25 application kernels representing a variety of behaviors common in the domain of HPC and scientific computing (see Section 6).

4.1. Performance Sensitivity Measurements

We execute the kernels and applications multiple times for multiple iterations across the entire design space of compute and memory configurations states described in Section 3.1. For each hardware configuration, we measure average execution time for each kernel across all the iterations. Sensitivity is computed for each hardware configuration. CU sensitivity is computed as the ratio of: i) relative change in execution times, to ii) relative change in number of active CUs. CU frequency and memory bandwidth are set to their maximum possible values in the hardware so that they are not the limiting factors. Sensitivities to CU frequency and memory bandwidth are similarly computed. Finally, the sensitivity to the number of CUs and CU frequency are aggregated into a single compute throughput sensitivity metric. The sensitivity models are then derived from these measurements as follows.

4.2. Performance Counter Correlation

Together with performance, we record an average of 50+ performance counters over all iterations of each kernel and application, resulting in one data point for every counter for each kernel at every hardware configuration. The counter selection is motivated by insights from Section 3.5. We normalize all counter values to a percentage of its maximum possible value in order to ensure proper weighted representation of all events in the training data. For a total of 25 kernels, this results in a total of 11250 vectors of performance counter values (25×450). We find that among multiple application kernels the performance counters vary quite a bit as expected. However, for the same kernel in an application with the same input set across multiple hardware configurations, there are generally only small variations around the nominal values. Therefore, each performance counter value for a kernel is replaced by its average value across all hardware configurations. This enables us to reduce the total training data set to 2000 points across all kernels. Each such vector is associated with its corresponding compute throughput sensitivity and memory bandwidth sensitivity.

Counter or Metric	Description
VALUUtilization	Percentage of active vector ALU threads in a wave, indicates branch divergence
MemUnitBusy	Percentage of total GPU time the memory fetch/read unit is active, including stalls and cache effects
MemUnitStalled	Percentage of total GPU time the memory fetch/read unit is stalled
WriteUnitStalled	Percentage of total GPU time memory write/store unit is stalled
NormVGPR	Number of general purpose vector registers used by the kernel, normalized by max 256
NormSGPR	Number of general purpose scalar registers used by the kernel, normalized by max 102
icActivity	Off-chip interconnect bus utilization between GPU L2 and DRAM
Compute-to-Memory Intensity (C-to-M Intensity)	Ratio of the time the vector ALU unit is busy processing active threads (VALUBusy*VALUUtilization) to the time the memory unit is busy (MemUnitBusy), normalized to 100

Table 2: Performance counters and metrics.

Bandwidth Sensitivity		Compute Sensitivity	
Counter or Metric	**Coefficient**	**Counter or Metric**	**Coefficient**
Intercept	-0.42	Intercept	0.06
VALUUtilization	0.003	C-to-M Intensity	0.007
WriteUnitStalled	0.011	NormVGPR	0.452
MemUnitBusy	0.01	NormSGPR	0.024
MemUnitStalled	-0.004		
icActivity	1.003		
NormVGPR	1.158		
NormSGPR	-0.731		

Table 3: Sensitivity model parameters.

4.3. Sensitivity Predictor Creation

Across the 2000 points, we perform a correlation analysis between measured sensitivities and performance counters across all kernels using linear regression. Coefficient values

greater than 0.5 or less than -0.5 are considered a strong positive or negative correlation, respectively [4]. From correlation analysis, we select a few counters to capture behaviors identified in Section 3.5 that have a substantive impact on sensitivities, as shown in Table 2. These are used to construct a linear regression model for compute throughput and memory bandwidth sensitivity. The correlation coefficient using this combination of metrics is 0.91 for compute throughput sensitivity and 0.96 for bandwidth sensitivity respectively. Accuracies of these predictors are discussed in Section 7.2. Table 3 represents the coefficients of the linear regression models. Two metrics are not directly available in hardware performance counters, and are calculated as follows.

$$icActivity = \frac{Read\_Write\_Mem\_BW}{Peak\_Mem\_BW}, \text{ where} \quad (1)$$

$$Peak\_Mem\_BW = Mem\_Frequency * Bus\_Width * \#Mem\_Channels * GDDR5\_Transfer\_Rate \quad (2)$$

To determine C-to-M intensity of an application online, we use the following metric:

$$C-to-M\ Intensity = \frac{\%\ time\ GPU\ is\ busy\ processing\ active\ ALU\ operations}{\%\ time\ GPU\ is\ busy\ processing\ memory\ operations)} = \frac{(VALUBusy * VALUUtilization)/100}{MemUnitBusy} \quad (3)$$

We believe principles of hardware balance and coordinated management are portable across platforms. Therefore, we expect the methodology is portable since most platforms provide similar classes of counters.

5. Harmonia: Two-Level Power Management

Based on the preceding analysis, we find that an effective approach to achieving hardware balance involves two steps: i) employing sensitivities to the hardware tunables to make larger adjustments to the hardware configurations, and ii) fine tuning the configurations based on performance feedback to further improve hardware balance. We refer to the former as coarse-grain (CG) tuning and the latter as fine-grain (FG) tuning. As the number of hardware power configurations grows in future processors we expect such coarse-fine schemes will be increasingly effective. Algorithm 1 specifies Harmonia.

5.1. Harmonia: Structure

Harmonia operates as a system software policy overlaid on top of the baseline HD7970 power management system. As described in Section 2, the baseline policy manages power to just the power states mentioned in Section 2.3. Our implementation is organized into: i) a monitoring block that samples the performance counters at application kernel boundaries, ii) a coarse-grain decision block CG that calculates memory bandwidth and compute throughput sensitivities based on Table 3 and brings the hardware configuration to the "vicinity" of the balance point, and iii) a fine-grain tuning block FG that fine-tunes configurations to further improve balance, based on real time performance feedback. Although the monitoring and decision blocks of Harmonia can operate at periodic small intervals, due to performance counter limitations in the current device, we monitor and calculate sensitivities at kernel boundaries and use each kernel's historical data from previous iterations to predict hardware configurations for the same kernel in the next iteration. For applications that use iterative convergence algorithms and invoke the entire application with multiple kernels multiple times, Harmonia records the last best hardware configuration for all kernels within that application. This state is the initial state for the subsequent iteration. Such iterative behaviors are quite common in HPC and scientific applications.

5.2. Harmonia: Algorithm

Within the CG block, all three tunables are concurrently adjusted in *SetCU-Freq-MemBW()*. Sensitivity is computed for each tunable using weighted linear equation per Table 3, and binned into three bins of high, medium, and low. Each bin is associated with a specific empirically fixed high, medium, or low value of the tunable sensitivity (i.e., core-frequency, CU, memory-BW). In our case, the three bins are set to <30%, 30%-70%, and >70%. The change in actual values of the hardware tunables is proportional to the sensitivity value. Periodic enforcement of hardware configurations can artificially change sensitivities and dampen natural workload behavior. To prevent this and isolate sensitivity changes due to workload from those due to changes in the hardware tunables, we only execute CG when there have been no changes in the hardware tunables prior to the sensitivity change.

Harmonia's FG block fine-tunes each of the hardware tunables based on performance feedback through the gradient of core utilization. The idea is to reduce power when the gradient is positive or zero and increase power when the gradient is negative so as to eventually settle at the balance point (minimum configuration with zero gradient). To prevent oscillation, the configuration is set to the last best state after a certain number of oscillations to enable convergence prior to the next workload phase. We found that changes in the *VALUBusy* performance counter (i.e., percentage of time processing vector-ALU instructions) are a good proxy for changes in "overall" performance. If sensitivity for any tunable does not change between two subsequent iterations, the FG step is invoked to change that tunable by one step-size at a time (core step=100MHz, memory BW step=30GB/s, CU step=4, defined in Section 3.1). All tunables can be fine-tuned concurrently. FG adjustments occur continuously as long as performance improves or stays same (as evidenced by changes in *VALUBusy*). If performance starts to degrade, FG isolates the responsible tunable and reverts it to previous value. The control-loop seeks to settle at the minimum value of the tunables minimizing power without hurting performance.

6. Experimental Setup

We use an AMD Radeon HD7970 discrete graphics card with 32 compute units as the baseline for all our experiments and analysis. The possible hardware configurations are provided in Section 2.3. In our analysis, there are 450 possible combi-

```
while TRUE do
    //Monitoring Loop
    //Online Sensitivity Computation Loop
    Compute Throughput Sensitivity = model1;
    Bandwidth Sensitivity = model2;
    Bin sensitivities to HIGH, MED, LOW;
    //Coarse-Grained Tuning (CG Block)
    if sensitivity changed then
        if CU or comp_freq or mem_freq changed in previous iterations then
            Revert_prev_decision(); //sensitivities artificially changed due to
            configuration change
        end
        else
            //Application phase change
            SetCU_Freq_MemBW(sensitivity_bin);
        end
    end
    else
        //Case of same sensitivities
        //Fine-Grained Tuning (FG Block)
        if VALUBusy gradient >= 0 then
            Decrement state; //CU, CU_Freq, or Mem_BW
        end
        else
            if VALUBusy gradient < 0 then
                Increment state;
                CountDithering();
                if dithering > max then
                    converge to last state with zero gradient;
                end
            end
        end
    end
    Run at config identified;
    Sleep.time(SAMPLING_INTERVAL);
end
```

Algorithm 1: Pseudo Code of Harmonia.

nations of the number of active CUs, compute frequency, and memory bus frequency as described in Section 3.1. When varying compute frequency, voltage is also scaled as noted in Table 1. When scaling memory bus frequency, voltage is fixed at the hardware default value due to platform constraints. All inactive CUs are power gated. Hardware performance counters are monitored using the GPU performance counter library CodeXL running in Red Hat Linux OS [10]. We implement Harmonia as a run-time system software policy by layering it on top of the baseline AMD HD7970 power-management system.

We select 14 applications with many kernels, covering a wide range of typical applications to reflect the needs of the HPC and scientific computing community. Targeting the HPC community and the ability to stress compute or memory motivated our selections. They include Exascale HPC proxy apps (*CoMD, XSBench, miniFE*) [5] [22], *Graph500* [37], B+Tree (*BPT*) [11], *CFD, LUD, SRAD* and *Streamcluster* from Rodinia [6, 7], and *Stencil, Sort, SPMV, MaxFlops* and *DeviceMemory* from SHOC [12]. We run each application multiple times and recorded the average to eliminate run-to-run variance in our hardware measurements.

We measure performance as the total execution time of the application running on the GPU. Power is profiled using a National Instruments data acquisition (DAQ) card (NI PCIe-6353), with a sampling frequency of 1KHz. Total GPU card power (GPUCardPwr) is measured at the PCI-e connector interface between the motherboard and the GPU card and it includes power of the GPU chip, its on-chip memory controller, DDR bus transceivers (PHYs), off-chip GDDR5 memory, fan, voltage regulators, and other miscellaneous components on the card. We also separately measure the GPU chip power (GPUPwr) which includes power of the GPU compute, integrated memory controller, but not the phys. Through detailed measurements and evaluation under idle conditions, we characterize the "rest of the card power" (OtherPwr) as power due to the fan, voltage regulators, board trace losses, and other minor discrete components. To ensure a constant OtherPwr, we fix the fan speed to the highest RPM at all times, independent of the workload. Based on these measurements, we derive memory power (MemPwr) as the power consumed by off-chip memory and DDR PHYs that are integrated within the GPU chip. Due to platform measurement constraints, memory controller power is not included in measured memory power, instead it is part of GPUPwr, but it accounts for only about 3% of the overall memory power in our case.

$$MemPwr = GPUCardPwr - GPUPwr - OtherPwr \quad (4)$$

7. Results

All results are obtained from commodity hardware and are normalized to the baseline HD7970 power management system discussed in Section 2.3. All averages represent the geometric mean across the applications. Finally, we also compare Harmonia with an oracle scheme optimized for ED^2 based on exhaustive online profiling of every iteration of each kernel across all of the 450 possible hardware configurations (see Section 3.1). While the oracle technique provides a useful basis for evaluation, it is impractical to implement.

7.1. Performance, Power, and Energy Efficiency

Figures 10 through 12 illustrate improvements in ED^2, energy, and power respectively relative to the baseline and the oracle. In addition, we also demonstrate the performance of just CG tuning. Harmonia is represented by the "FG+CG" bars. Due to the consistent availability of thermal headroom, the baseline power management always runs at the boost frequency of 1GHz for all applications. We show two geometric means to ensure results are not skewed by the stress benchmarks *MaxFlops* and *DeviceMemory*, which represent extreme cases of compute and memory limiting respectively. Geomean_2, which is the last set of bars, excludes those two stress benchmarks. Harmonia realizes an average ED^2 improvement of 12% compared to the baseline, with up to 36% savings in *BPT*. Of this 12% ED^2 savings, about 6% is due to CG tuning, with the remaining from the fine-grain tuning. In addition, Harmonia is typically within 3% of the oracle. Interestingly, we observe that the energy savings is almost identical between the CG and FG+CG schemes, with a contribution of only 2% coming from the FG loop. However, FG tuning is important for preserving performance as described next.

We observe an average power savings of 12% across the entire GPU card, with a maximum savings of 19% for *Stencil*. During application phases less sensitive to memory bandwidth such as *EAM_Force_1* in *CoMD*, reducing memory bus frequency just enough without increasing memory-related stalling and exposing memory access latency results in reduc-

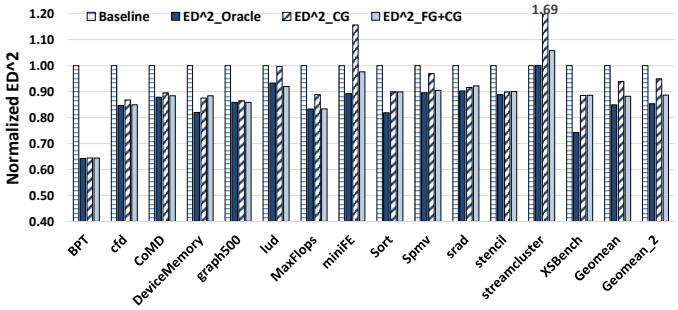

Figure 10: Overall combined performance and energy gain from Harmonia, using the ED^2 metric.

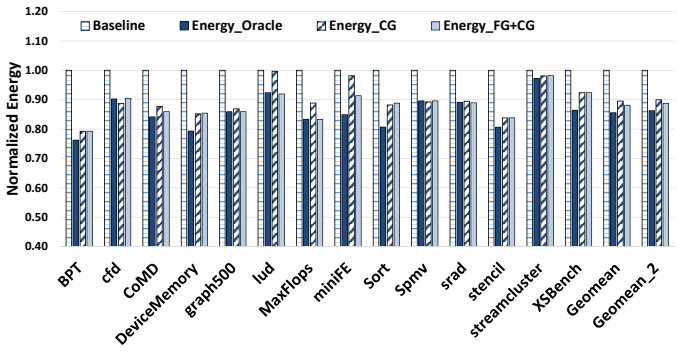

Figure 11: Overall energy gain from Harmonia.

tion of memory bus power and thereby savings of the overall board power–GPUCardPwr. Notice that more memory power saving would be possible if HD7970's memory interface supports multiple voltages. On the other hand, *AdvanceVelocity* in *CoMD* is memory intensive with moderate compute demands and Harmonia finds the balance points by reducing compute power without performance loss. Similarly, due to poor thread-level parallelism (kernel occupancy of 30%) in *BottomScan*, the main kernel in *Sort*, the memory bus frequency can be reduced down to 475MHz without hurting performance with a 12% overall GPU card power savings. In some cases, power savings can be a bit worse in FG+CG than in CG, as Harmonia puts more emphasis on performance.

In Figure 13 we see an average loss in performance of 0.36% across all the applications using Harmonia (FG+CG) excluding *MaxFlops* and *DeviceMemory*, with up to 3.6% maximum slow-down in *Streamcluster* due to the edge effect of sensitivity binning (i.e., narrowly missing the HIGH bin). This illustrates the efficacy of Harmonia in optimizing energy efficiency under performance constraints by pushing the hardware to operate at its balance for each application's kernel. We also note that employing CG tuning alone results in an average performance loss of 2.2% compared to the baseline, with up to 27% maximum slow-down for *Streamcluster*. This is due to the lack of any performance feedback in CG tuning. Thus, while the use of CG tuning alone achieves energy savings comparable to Harmonia, the performance-driven FG tuning loop ensures much better performance across all applications and avoids outliers resulting in better overall ED^2.

There are three applications that are worth noting here. They are *BPT, CFD*, and *XSBench*. These applications see

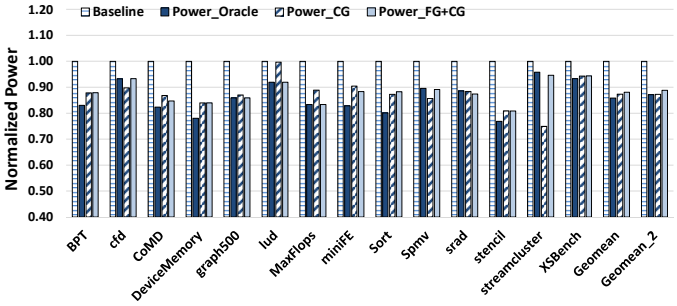

Figure 12: Overall power savings from Harmonia.

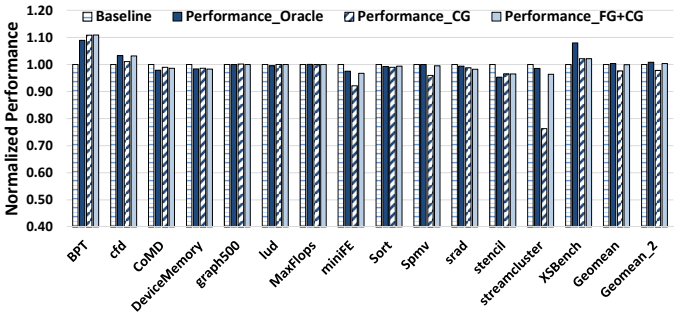

Figure 13: Overall performance from Harmonia.

an *improvement* in performance with Harmonia. *BPT* sees an 11% performance gain, while *CFD* and *XSBench* each realize 3% performance improvement. In the baseline hardware configuration, we observe heavy cache thrashing and pollution accompanied by significant memory divergence. Thus, lowering the number of active CUs via power gating also improves performance by reducing interference in the L2 cache. Harmonia captures the optimal compute to memory balance point via the sensitivity to CU count for these applications.

7.2. Adaptation Behavior

In this section, we explain the adaptation behavior of Harmonia in response to workload changes.

Intra-kernel Phase Changes: Figure 14 illustrates the time-varying workload behavior of the main kernel *BottomStepup* in *Graph500*. The Y-axis indicates the total number of compute instructions (VALUInsts), memory reads (VFetchInsts) and memory writes (VWriteInsts) executed in eight successive iterations, each iteration lasting anywhere from 0.9 to 5.6 seconds. This kernel is performing a breadth-first search. Note that the raw total number of instructions across iterations can vary significantly.

The memory fetch unit is active anywhere from 40% to

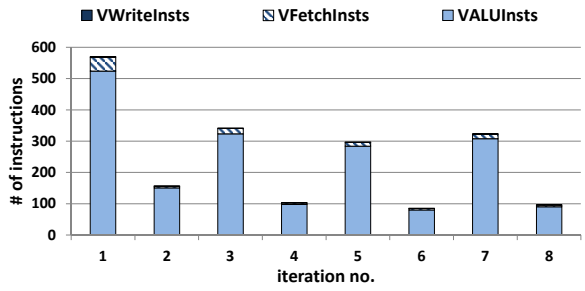

Figure 14: Behavior of *Graph500.BottomStepUp* over time.

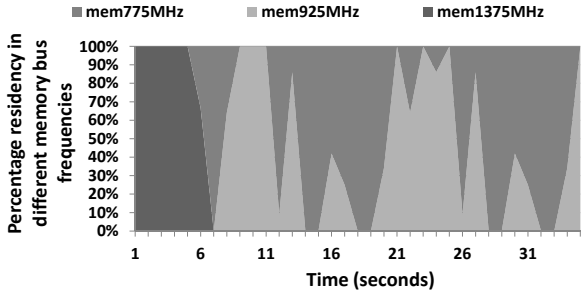

Figure 15: Memory bus frequency residency changes as time progresses in *Graph500.BottomStepUp*.

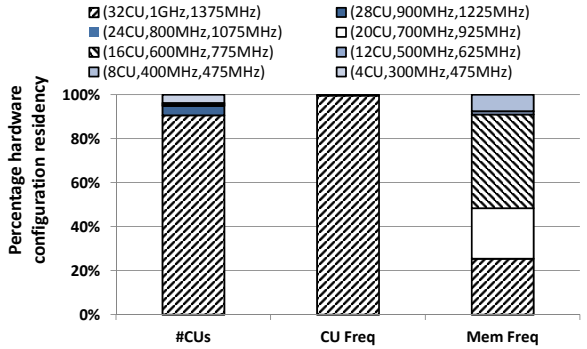

Figure 16: Residency of the hardware tunables in *Graph500*.

80% of the total kernel execution time. The compute sensitivity is high 95% of the time and branch divergence is significant. As a result Harmonia mostly utilizes all 32 CUs and 1GHz compute frequency to speed up execution of threads serialized by branch divergence. However, bandwidth sensitivity changes frequently between medium and low as the predictor adapts to input argument changes and consequent changes in demand for memory bandwidth. Thus, through CG and FG tuning, memory frequency dithers between 925MHz and 775MHz. Figure 15 shows the distribution of time spent at the different memory bus frequencies in Harmonia over the kernel's entire execution.

Inter-kernel Phase Changes: We observe that the ops/byte value of *Graph500* varies from 0.64 to 264. Figure 16 shows the fraction of time each hardware tunable spends in each power state as Harmonia moves the hardware towards the right balance point. For this application due to high branch divergence, Harmonia tunes to the maximum compute frequency (single state in CUFreq column). This is accompanied by tuning of the CU count and memory bandwidth that reduces power. The #CUs column shows that about 90% of the time 32 CUs are used; the remaining time is spent in dithering between 4, 8, 12, and 16 CUs based on time-varying ops/byte. The memory bus frequency varies between 1375MHz (25% of the time), 925MHz (23%), 775MHz (42%), and 475MHz (8%) as the operational intensity of the three kernels in *Graph500* varies from 0.64 ops/byte to bursts of 264 ops/byte.

Coordinated Power Sharing: Figure 17 shows the GPU and memory power consumption across a subset of the applications with both baseline and Harmonia (HM), relative to the measured total power for GPU and memory. Here the total power is normalized with respect to the baseline. Power due to remaining elements on the board are not shown since they are roughly constant. We observe that out of the average 12% power savings, 64% of the savings comes from varying the GPU compute configuration. The remaining 36% comes from changing the memory bus frequencies. We believe that it is feasible to achieve far more power savings from memory configuration changes if voltage scaling is applied while lowering bus speeds. In our current setup we are not able to scale voltage of the memory bus interface. Harmonia seeks a balance of core and memory power, i.e. just enough core power is expended to utilize all requested memory bandwidth and vice versa.

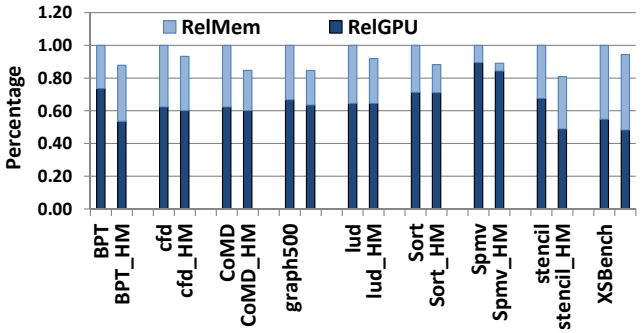

Figure 17: Relative GPU and memory power consumption.

Another interesting observation is that most often Harmonia adjusts CU counts and memory bus frequencies rather than the full range of compute frequencies. This behavior is consistent across all applications. In fact compute frequency and voltage scaling alone achieve only an average ED^2 gain of 3% with a 1% performance loss compared to the baseline. The reason is two-fold: i) parallel execution and data movement demands are inherent to the application and govern demanded ops/byte values which vary widely across applications, thus making available hardware resources in excess of these demands is not helpful, and ii) as explained in Section 3.5, architectural clock domain crossings reduce opportunities for compute frequency to improve energy efficiency for memory intensive applications.

Algorithm Convergence and Relative Impact of CG versus FG Tuning: Figure 18 shows the relative contributions from CG and FG tuning for energy efficiency improvement across a subset of applications. In most applications CG tuning requires only one iteration. Even in applications with a small number of iterations (insufficient for feedback driven FG tuning), CG is very effective in rapidly reaching a lower power operation point often in a single iteration. An example is *XSBench* which executes only 2 iterations for each of its kernels. Even here, Harmonia is able to save 4% overall GPU card power while improving overall application performance by 2%, resulting in 9% energy efficiency gain. However, in certain cases such as *LUD, SPMV*, due to prediction outliers or lack of performance feedback, CG can leave out additional power savings opportunities or degrade performance. In such cases, FG tuning plays a crucial role. The FG step typically takes an additional 3 to 4 iterations to converge. In HPC applications, many kernels represent iterative computations that

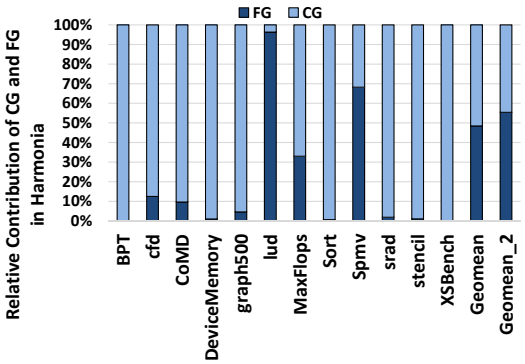

Figure 18: Relative contributions of CG versus FG in Harmonia.

typically execute several times to converge with minimal algorithmic error. For such kernels, the overhead of FG tuning is amortized over successive kernel invocations. Therefore, both steps are necessary in order to have Harmonia cover a broad range of workloads.

Sensitivity Predictors: The prediction errors between measured and estimated bandwidth and compute sensitivities are 3.03% and 5.71% respectively across all the applications used in this study. Since our goal is to develop simple, effective, and practical sensitivity predictors that can be easily implemented in hardware, we find that simple linear regression based sensitivity models, such as the ones proposed in this paper combined with an effective binning methodology can significantly help improve the accuracy of the predictors.

7.3. Summary of Key Insights

In this section, we summarize our main results and insights:

1. Compute and memory behavior are fundamentally performance coupled. Optimizing only compute or memory behavior has limited benefits. It is necessary to balance the time and energy costs of compute and memory to improve energy efficiency with minimal performance loss.
2. Scaling parallelism (number of active CUs) and memory bandwidth is more effective than scaling CU frequency since it has a greater impact on ops/byte behaviors. Note that modern systems rely primarily on scaling compute frequency for energy efficiency gains.
3. Clock domain crossings and interconnect sizing have a non-trivial impact on energy efficiency.
4. Feedback driven fine-grained adjustments are effective in correcting coarse-grain tuning mispredictions or longer term changes in learned behaviors.
5. Improving energy efficiency can lead to improvements in execution time due to reduction of interference in shared resources (e.g., cache or interconnect).
6. With advanced packaging technologies, compute and memory will share tighter package power envelopes (e.g., compute with stacked memory) [43, 26, 38]. Coordinated power management and the concept of hardware balance will become increasingly important in such systems.

8. Related Work

In this section, we survey prior studies that consider interactions among CPU, GPU, and memory. The authors in [2, 31, 34, 50] analyze interactions between CPU-GPU, and propose a power-efficient way of work distribution between the CPU and GPU for throughput-computing applications. [32, 33] investigate the impact of GPU core and frequency scaling and propose a GPU power-model to study power savings from core DVFS. Both use simulators. [44] employ whole-chip thermal-based power management. "Energy credits" are allocated to the CPU and/or GPU with awareness of the dependency of performance between the two. [41] take a step further by also considering thermal coupling between the CPU and GPU, in addition to performance coupling. There are also many existing studies investigating main memory power management in CPU-memory systems [14, 13, 17]. For example, Deng et al. in Memscale [17] apply DVFS to memory controllers and DFS to memory channels and DRAM devices, evaluated using a simulation framework. [13] propose DVFS for main memory and presents evaluations on real hardware. [14] allocate a power cap to main memory with the aid of a runtime DRAM power model. In addition, several research works have focused on DVFS in cores for power and thermal management [24, 25, 27].

A few prior works also look at coordinated power management between CPU and main memory. [48] is an insightful approach for multithreaded CPUs, but not for thousands of fine-grain bulk-synchronous threads. The authors in Coscale[15] propose runtime techniques to minimize total system energy within a performance constraint for a multi-CPU system, using a simulation framework. Deng et-al in Multiscale [16] attempt to reduce system energy by applying coordinated DFVS across multiple memory controllers (MCs), based on the observation of skewed traffic across MCs in multicore server processors. Chen et al. [8] study power capping for servers and control both processor and memory power. Diniz et al. [18] allocate a power cap to memory in a CPU-memory system. Other related work examines policies [20] and analytic models [23] while we note that in the HPC community, there has been a considerable effort in tuning and managing power in CPU-memory architectures [30, 49, 40, 45, 46].

Our work is distinctive in its focus on dynamically monitoring and managing GPU-memory interactions, which are quite distinct and therefore merit distinct solutions. Further, unlike many of these efforts, we seek to concurrently minimize performance impact rather than trade performance for improvements in energy efficiency.

9. Conclusions

This paper applies the notion of hardware balance to the development of a practical scheme for the coordinated management of compute and memory power in a high performance discrete GPU platform. By tracking the time-varying relative compute and memory demands of applications, the corresponding hardware power configurations of the core and memory system can be set to reduce overall platform power and thereby improve energy efficiency with minimal compromises in performance. In the future, we plan to expand this work to manage an integrated CPU-GPU-memory system with stacked memory architectures.

References

[1] AMD, "PowerTune Technology whitepaper, 2010."

[2] M. Arora, S. Nath, S. Mazumdar, S. Baden, and D. Tullsen, "Redefining the Role of the CPU in the Era of CPU-GPU Integration," *IEEE Micro*, 2012.

[3] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," *Technical Report UCB/EECS-183.2006*, 2006.

[4] W. L. Bircher, M. Valluri, J. Law, and L. John, "Runtime Identification of Microprocessor Energy Saving Opportunities," in *International Symp. on Low Power Electronics and Design (ISLPED)*, 2005.

[5] W. Brown, P. Wang, S. Plimpton, and A. Tharrington, "Implementing Molecular Dynamics on Hybrid High Performance Computers—Short Range Forces," *Compute Physics Communications*, 2011.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE Intl. Symp. on Workload Characterization*, 2009.

[7] S. Che, J. Sheaffer, M. Boyer, L. Szafaryn, and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *IEEE Intl. Symp. on Workload Characterization*, 2011.

[8] M. Chen, X. Wang, and X. Li, "Coordinating Processor and Main Memory for Efficient Server Power Control," in *International Conference on Supercomputing (ICS)*, 2011.

[9] J. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A Roofline Model of Energy," in *IEEE International Distributed Process Symposium*, 2013.

[10] CodeXL, "http://developer.amd.com/tools-and-sdks/heterogeneous-computing/codexl/."

[11] M. Daga and M. Nutter, "Exploiting Coarse-grained Parallelism in B+ Tree Searches on APUs," in *Workshop on Irregular Applications, Architectures and Algorithms (IA3)*, 2012.

[12] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter, "The Scalable Heterogeneous Computing (SHOC) Benchmarking Suite," in *Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU)*, 2010.

[13] H. David, C. Fallin, E. Gorbatov, U. Hanebutte, and O. Mutlu, "Memory Power Management vis Dynamic Voltage/Frequency Scaling," in *International Conference on Autonomous Computing (ICAC)*, 2011.

[14] H. David, E. Gorbatov, U. Hanebutte, K. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2010.

[15] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini, "CoScale: Coordinating CPU and Memory System DVFS in Server Systems," in *International Symposium on Microarchitecture (MICRO)*, 2012.

[16] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini, "MultiScale: Memory System DVFS with Multiple Memory Controllers," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2012.

[17] Q. Deng, D. Meisner, L. Ramos, T. Wenisch, and R. Bianchini, "MemScale: Active Low-Power Modes for Main Memory," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[18] B. Diniz, D. Guedez, W. Meira, and R. Bianchini, "Limiting the Power Consumption of Main Memory," in *International Symposium on Computer Architecture (ISCA)*, 2007.

[19] Elpida, "http://www.elpida.com/en/news/2011/06-27.html."

[20] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems," in *International Conference on Supercomputing (ICS)*, 2005.

[21] Green500 List, "http://www.green500.org."

[22] M. Heroux, D. Doerfler, P. Crozier, J. Willenbring, H. Edwards, A. Williams, M. Rajan, E. Keiter, H. Thornquist, and R. Numrich, "Improving Performance via Mini-applications," *Sandia Report, SAND2009-5574*, 2009.

[23] S. Hong and H. Kim, "An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness," in *International Symposium on Computer Architecture (ISCA)*, 2009.

[24] C. Hsu and W. Feng, "Effective Dynamic Voltage Scaling through CPU-Boundedness Detection," *Lec. Notes in Computer Science*, 2004.

[25] W. Huang, M. Stan, K. Sankaranarayanan, R. Ribando, and K. Skadron, "Many-core Design from a Thermal Perspective," in *Design Automation Conference (DAC)*, 2008.

[26] JEDEC Wide I/O, "http://www.jedec.org/news/pressreleases/jedecpublishes-breakthrough-standard-wide-io-mobile-dram, jan 2012."

[27] S. Kaxiras and M. Martonosi, "Computer Architecture Techniques for Power Efficiency," *Synth. Lec. on Computer Architecture*, 2008.

[28] S. Keckler, W. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *IEEE Micro*, 2011.

[29] G. Kestor, R. Gioiosa, D. Kerbyson, and A. Hoisie, "Quantifying the Energy Cost of Data Movement in Scientific Applications," in *International Symposium on Workload Characterization (IISWC)*, 2013.

[30] J. Laros, K. Pedretti, S. Kelly, W. Shu, and C. Vaughan, "Energy Based Performance Tuning for Large Scale High Performance Computing Systems," in *Symp. on High-Performance Computing*, 2012.

[31] J. Lee and H. Kim, "TAP: A TLP-Aware Cache Management Policy for a CPU-GPU Heterogeneous Architecture," in *International Conference on High-Performance Computer Architecture (HPCA)*, 2012.

[32] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim, "Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2011.

[33] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *International Symposium on Computer Architecture (ISCA)*, 2013.

[34] C. Luk, S. Hong, and H. Kim, "Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping," in *International Symposium on Microarchitecture (MICRO)*, 2009.

[35] M. Mantor and M. Houston, "AMD Graphics Core Next," in *AMD Fusion Developer Summit*, 2011.

[36] A. McLaughlin, I. Paul, J. Greathouse, S. Manne, and S. Yalamanchili, "A Power Characterization and Management of GPU Graph Traversal," in *Workshop on Architectures and Systems for Big Data*, 2014.

[37] R. Murphy, K. Wheeler, B. Barett, and J. Ang, "Introduing the Graph500," *Cray User's Group (CUG)*, 2010.

[38] Online, "http://www.anandtech.com/show/8217/intels-knights-landing-coprocessor-detailed."

[39] Online, "http://www.techspot.com/news/52003-future-nvidia-volta-gpu-has-stacked-dram-offers-1tb-s-bandwidth.html, march 2013."

[40] S. Pakin, C. Storlie, M. Lang, R. Fields, E. Romero, C. Idler, S. Michalak, H. Greeberg, J. Loncaric, R. Rheinheimer, G. Grider, and J. Wendelberger, "Power Usage of Production Supercomputers and Production Workloads," in *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2012.

[41] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili, "Cooperative Boost: Needy vs. Greedy Power Management," in *International Symposium on Computer Architecture (ISCA)*, 2013.

[42] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili, "Coordinated Energy Management in Heterogeneous Processors," in *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2013.

[43] J. Pawlowski, "Hybrid Memory Cube (HMC)," in *HotChips*, 2011.

[44] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissman, "Power Management Architectures of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, 2012.

[45] B. Rountree, D. Lowenthal, B. de Supinski, M. Schulz, V. Freeh, and T. Bletsch, "Adagio: Making DVS Practical for Complex HPC Applications," in *International Conference on Supercomputing (ICS)*, 2009.

[46] B. Rountree, D. Lowenthal, S. Funk, V. Freeh, B. de Supinski, and M. Schulz, "Bounding Energy Consumption in Large-Scale MPI Programs," in *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2007.

[47] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *International Conference on High Performance Computing for Computational Science*, 2010.

[48] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das, "PEPON: performance-aware hierarchical power budgeting for NoC based multicores," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[49] A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely, "Autotuning for Energy Usage in Scientific Applications," in *International Conference on Parallel Processing (Euro-Par)*, 2011.

[50] H. Wang, V. Sathish, R. Singh, M. Schulte, and N. Kim, "Worload and Power Budgest Partitioning for Single Chip Heterogeneous Processors," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[51] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, 2009.